

Introduction

Soft errors occur when high-energy charged particles alter the stored charge in a memory cell in an electronic circuit. The phenomenon first became an issue in DRAM, requiring error detection and correction for large memory systems in high-reliability applications. As device geometries have continued to shrink, the probability of soft errors in SRAM has become significant for some systems. Designers are using a variety of approaches to minimize the effects of soft errors on system behavior.

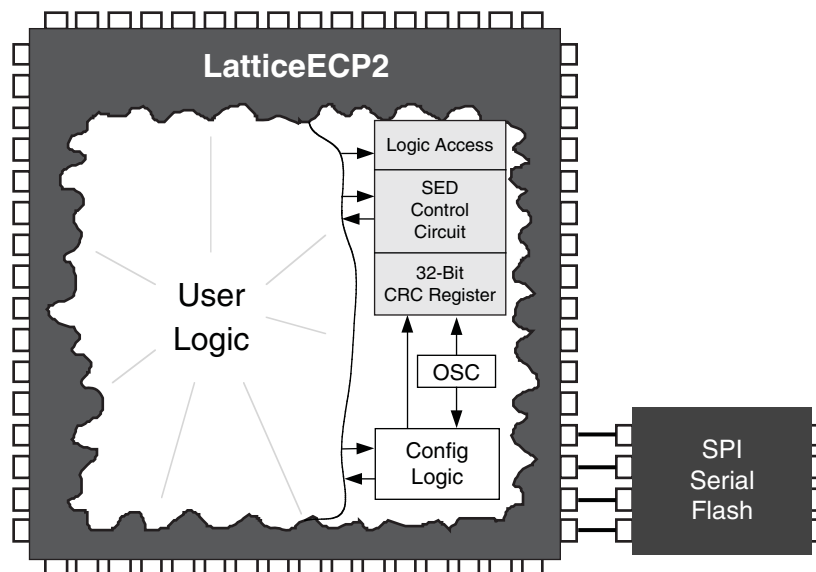
SRAM-based FPGAs store logic configuration data in SRAM cells. As the number and density of SRAM cells in an FPGA increase, the probability that a soft error will alter the programmed logical behavior of the system increases. A number of approaches have been taken to address this issue, but most involve Intellectual Property (IP) cores that the user instantiates into the logic of their design, using valuable resources and possibly affecting design performance.

This document describes the hardware based soft error detect (SED) approach taken by Lattice Semiconductor for the new LatticeECP2™ FPGAs.

SED Overview

The SED hardware in the LatticeECP2 devices consists of an access point to FPGA configuration memory, a controller circuit, and a 32-bit register to store the CRC for a given bitstream (see Figure 1). The SED hardware reads serial data from the FPGA's configuration memory and calculates a CRC. The data that is read, and the CRC that is calculated, does not include EBR memory or PFUs used as RAM. The calculated CRC is then compared with the expected CRC that was stored in the 32-bit register. If the CRC values match it indicates that there has been no configuration memory corruption, but if the values differ an error signal is generated. SED checking does not impact the performance or operation of the user logic.

Figure 1. System Block Diagram



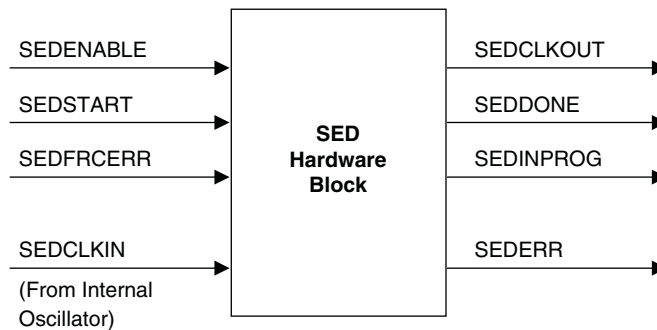
Note that the calculated CRC is based on the particular arrangement of configuration memory for a particular design. Consequently, the expected CRC results cannot be specified until after the design is placed and routed. The ispLEVER[®] bitstream generation software analyzes the configuration of a placed and routed design and updates the 32-bit SED CRC register contents during bitstream generation.

The following sections describe the LatticeECP2 SED implementation and flow, along with some sample code to get started with.

Hardware Description

As shown in Figure 2, the LatticeECP2 SED hardware has several inputs and outputs that allow the user to control, and monitor, SED behavior.

Figure 2. Signal Block Diagram



Signal Description

Table 1. SED Signal Description

Signal Name	Direction	Active	Description
SEDCLKIN	Input	N/A	Clock
SEDENABLE	Input	High	SED enable
SEDCLKOUT	Output	N/A	Output clock
SEDSTART	Input	High	Start SED cycle
SEDINPROG	Output	High	SED cycle is in progress
SEDDONE	Output	High	SED cycle is complete
SEDFRCERR	Input	High	Force an SED error flag
SEDERR	Output	High	SED error flag

SEDCLKIN

Clock input to the SED hardware.

This clock is derived from the LatticeECP2's on-chip oscillator. The on-chip oscillator's output goes through a divider to create MCCLK. MCCLK goes through another divider to create SEDCLKIN.

MCCLK defaults to 2.5 MHz, but this can be modified using the MCCLK_FREQ global preference in ispLEVER's pre-map Design Planner (see Lattice technical note TN1108, *LatticeECP2 sysCONFIG Usage Guide*, for possible values of MCCLK).

The divider for SEDCLKIN can be set to 1, 2, 4, 8, 16, 32, 64, 128, or 256. The default is 1, so the default SEDCLKIN frequency is 2.5 MHz. The divider value can be set using a parameter, see the example code at the end of this document.

SEDENABLE

Active high input to the SED hardware, sampled on the rising edge of SEDCLKIN.

Table 2. SEDENABLE

State	Description
1	Enables output of SEDCLKOUT, arms SED hardware.
0	Aborts SED and forces all SED hardware outputs low.

SEDCLKOUT

Gated version of SEDCLKIN, SEDCLKOUT is gated by SEDENABLE.

SEDSTART

Active high input to the SED hardware, sampled on the rising edge of SEDCLKIN.

Table 3. SEDSTART

State	Description
1	Start error detection. Must be high a minimum of one SEDCLKIN period.
0	No action.

SEDFRCERR

Active high input to the SED hardware, sampled on the rising edge of SEDCLKIN.

Table 4. SEDFRCERR

State	Description
1	Forces SEDERR high, simulating an SED error.
0	No action.

SEDINPROG

Active high output from the SED hardware, clocked out on the rising edge of SEDCLKOUT.

Table 5. SEDINPROG

State	Description
1	SED checking is in progress, goes high on the clock following SED-START high.
0	SED checking is not active.

SEDDONE

Active high output from the SED hardware, clocked out on the rising edge of SEDCLKOUT.

Table 6. SEDDONE

State	Description
1	SED checking is complete. Reset by a high on SEDSTART or a low on SEDENABLE.
0	SED checking is not complete.

SEDERR

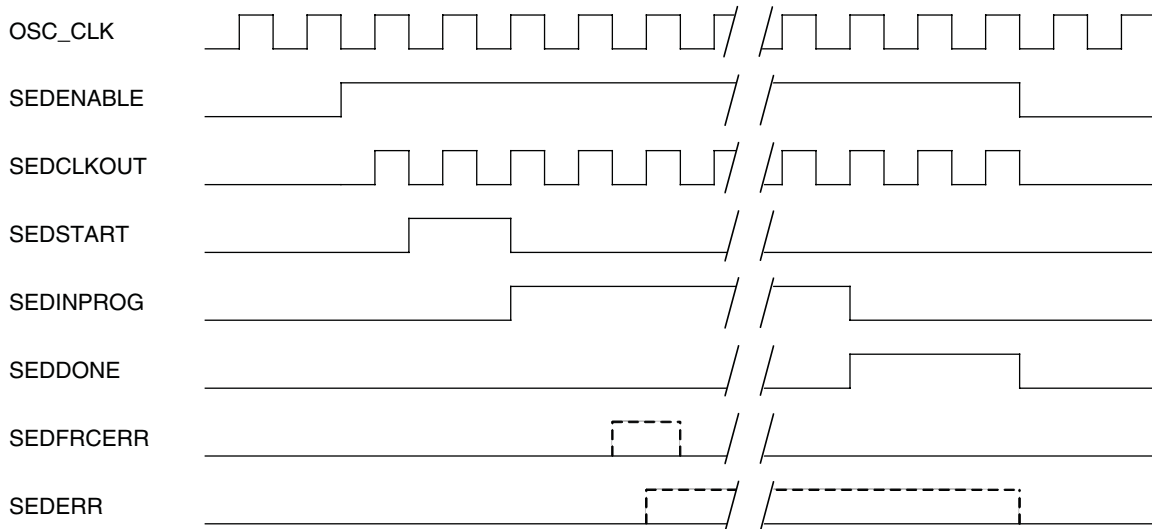
Active high output from the SED hardware, clocked out on the rising edge of SEDCLKOUT.

Table 7. SEDERR

State	Description
1	SED has detected an error. Reset by SEDENABLE going low.
0	SED has not detected an error.

SED Flow

Figure 3. Timing Diagram

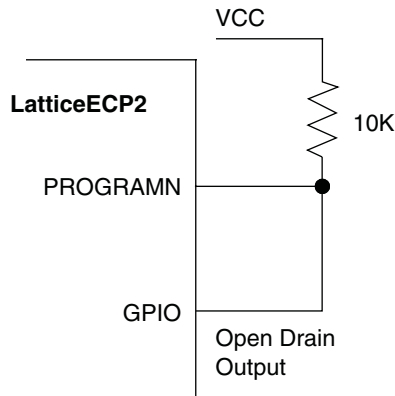


The general SED flow is as follows.

1. User logic sets SEDENABLE high. This signal may be tied high if desired.
2. User logic sets SEDSTART high. SEDINPROG goes high. If SEDDONE is already high it is driven low. SEDSTART may be tied high to enable continuous SED checking.
3. SED starts reading back data from the configuration SRAM.
4. SED finishes checking. SEDERR is updated, SEDINPROG goes low, and SEDDONE goes high.
5. If SEDERR is driven high there are only two ways to reset it, drive SEDENABLE low or reconfigure the FPGA.

The user has two choices when an error is detected, ignore the error, and possibly log it, or reconfigure the FPGA. Reconfiguration can be accomplished by driving the PROGRAMN pin low; this can be done with external logic or by wiring one of the FPGA's general purpose I/Os to the PROGRAMN pin and toggling the pin with user logic, perhaps something as simple as inverting SEDERR. If a general purpose I/O is tied to PROGRAMN it is recommended that the I/O Type be set to open drain and an external pull-up resistor be connected to the pin.

Figure 4. Example Schematic



SED Run Time

The amount of time needed to perform an SED check depends on the density of the device and the frequency of SEDCLKIN. There will also be some overhead time for calculation, but it is fairly short in comparison. An approximation of the time required can be found by using the following formula:

$$\text{Maxbits} / \text{SEDCLKIN} = \text{Time}$$

Maxbits is in mega-bits and depends on the density of the FPGA (see Table 8). SEDCLKIN is frequency in MHz. Time is in seconds

For example, if the design is using a LatticeECP2 with 50K look-up tables, and the SEDCLKIN is the default 2.5 MHz:

$$9.4 \text{ Mbits} / 2.5 \text{ MHz} = 3.76 \text{ seconds}$$

In this example, SED checking will take approximately 3.76 seconds. Remember that this happens in the background and does not affect user logic performance.

Note that the internal oscillator used to generate SEDCLKIN can vary by $\pm 30\%$.

Table 8. SED Run Time

Density	Bitstream Size (Mb)	Run Time ¹ (seconds)
ECP2-6	1.5	0.6
ECP2-12	2.9	1.16
ECP2-20	4.5	1.68
ECP2-35	6.3	2.52
ECP2-50	9.4	3.76
ECP2-70	13.3	5.32

1. Based on SEDCLKIN = 2.5 MHz.

Sample Code

The following simple example code shows how to instantiate the SED. In the example the SED is always on and always running, and the outputs of the SED hardware have been routed to FPGA output pins.

Note that the SEDAA primitive is part of ispLEVER 6.0 or later.

VHDL Example

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity example is
    port (
        Sed_Done      : out std_logic;
        Sed_In_Prog   : out std_logic;
        Sed_Clk_out    : out std_logic;
        Sed_out        : out std_logic);
end;

architecture behavioral of example is

    component SEDAA -- SED component
        generic (OSC_DIV : string := "1"); -- set SEDCLKIN divider
        port (
            SEDENABLE    : in std_logic;
            SEDSTART      : in std_logic;
            SEDFRCERR     : in std_logic;
            SEDERR        : out std_logic;
            SEDDONE       : out std_logic;
            SEDINPROG     : out std_logic;
            SEDCLKOUT     : out std_logic) ;
    end component;

    begin

        isnt1: SEDAA
            generic map (OSC_DIV=> "1")
            port map (
                SEDENABLE    => '1',    -- tied high
                SEDSTART      => '1',    -- tied high
                SEDFRCERR     => '0',    -- tied low
                SEDERR        => Sed_out, -- wired to an output
                SEDDONE       => Sed_Done, -- wired to an output
                SEDINPROG     => Sed_In_Prog, -- wired to an output
                SEDCLKOUT     => Sed_Clk_out ); -- wired to an output

    end behavioral ;
```

Verilog Example

```
module example (
    Sed_Done,
    Sed_In_Prog,
    Sed_Clk_out,
    Sed_out) ;

output Sed_Done;
output Sed_In_Prog;
output Sed_Clk_out;
output Sed_out;

assign V_hi = 1'b1;
assign V_lo = 1'b0;

parameter OSC_DIV = "1"; // set SEDCLKIN divider

    SEDAA sed_ip (
        .SEDENABLE(V_hi), // always high
        .SEDSTART(V_hi), // always high
        .SEDFRCERR(V_lo), // always low
        .SEDERR(Sed_out), // wired to an output
        .SEDDONE(Sed_Done), // wired to an output
        .SEDINPROG(Sed_In_Prog), // wired to an output
        .SEDCLKOUT(Sed_Clk_out)); // wired to an output

endmodule
```

Technical Support Assistance

Hotline: 1-800-LATTICE (North America)
+1-503-268-8001 (Outside North America)
e-mail: techsupport@latticesemi.com
Internet: www.latticesemi.com